MAXIMALLY NEGATIVE SIGNED FRACTIONAL NUMBER MULTIPLICATION

BACKGROUND OF THE INVENTION

Field of the Invention:

The present invention relates to systems and methods for instruction processing and, more particularly, to systems and methods for performing multiplication processing of two maximally negative signed fractional numbers.

Description of Prior Art:

Processors, including microprocessors, digital signal processors and microcontrollers, operate by running software programs that are embodied in one or more series of instructions stored in a memory. The processors run the software by fetching instructions from the series of instructions, decoding the instructions and executing the instructions. Processors, including digital signal processors, are conventionally adept at processing instructions that perform mathematical computations on positive fractional numbers specified as a data word. For example, some processors are adept at performing multiplicative operations, such as a 16-bit positive fractional number multiplied by another 16-bit fractional number. In general, multiplicative operations using 16-bit positive and negative fractional numbers produce a 32-bit result. The multiplication of two maximally negative 16-bit numbers produces a 33-bit result. The additional bit is required to represent the integer portion of the result. This type of multiplication employing two maximally negative fractional numbers requires an additional bit to represent the result of multiplying the two multiplied maximally negative 16-bit fractional numbers as well as a 17-bit DSP

multiplier to produce the result. The utilization of a 17-bit DSP multiplier in a processor is expensive, while the utilization of a 16-bit DSP multiplier produces inaccurate results.

There is a need for a new method of multiplying two maximally negative fractional numbers using a 16-bit DSP multiplier to produce a 32-bit result. There is a further need for a new method of producing a result of two multiplied maximally negative fractional numbers represented correctly with 32-bits. There is also a need for a new method of identifying when two maximally negative fractional numbers are multiplied.

SUMMARY OF THE INVENTION

According to embodiments of the present invention, methods and processors for multiplying two maximally negative fractional numbers to produce a 32-bit result are provided. This type of multiplication may be executed using a 16-bit DSP multiplier and produce a 32-bit result. The identification of a multiplication operation employing two maximally negative 16-bit fractional numbers enables manipulation of processing to correct a maximally negative result and produce a maximally positive result. Negate logic with a control block examines results produced by the 16-bit DSP multiplier and determines whether there are a combination of bits signifying the multiplication of two maximally negative 16-bit fractional numbers. The determination of the required bit combination initiates negate processing for correcting the results. This type of multiplication operation utilizes of a 16-bit DSP multiplier to produces accurate 32-bit results when the multiplication of two maximally negative fractional number occurs as well as reduces the overall cost of the processor.

According to an embodiment of the present invention, a method of multiplying two maximally negative fractional numbers to produce a 32-bit result includes fetching operands from a source location and performing a multiplication operation on the operands. The method also includes detecting that a result output of the multiplication operation corresponds to a maximally negative result. A maximally negative result indicates that the operands are two maximally negative fractional numbers. The method also includes correcting the result output to produce a maximally positive result output.

According to an embodiment of the present invention, a method of multiplying two maximally negative fractional numbers to produce a 32-bit result includes examining bits in a set of bits representing the result output to determining that the bits in the set of bits representing the result have a particular bit combination. The bit of particular importance include the thirtieth and thirty-first bits in the set of bits representing the result output and have a value of one and zero respectively.

According to an embodiment of the present invention, a method of multiplying two maximally negative fractional numbers to produce a 32-bit result includes generating a control signal. The control signal modifies a negate signal for controlling the performance of a two's compliment on the result output to produce a maximally positive result output. The maximally positive result output is accumulated in an accumulator.

According to an embodiment of the present invention, a method of multiplying two maximally negative fractional numbers to produce a 32-bit result includes fractionally aligning the result output. Fractional alignment includes shifting a set of bits representing the result output to the left by one bit to discard the most significant bit of the set of bits

representing the result output and insert a zero as the least significant bit of the set of bits representing the result output.

According to an embodiment of the present invention, a method of multiplying two maximally negative fractional numbers to produce a 32-bit result includes sign extending the output result. Sign extension includes extending the result output from a 32-bit result to a 40-bit result.

BRIEF DESCRIPTION OF THE DRAWINGS

The above described features and advantages of the present invention will be more fully appreciated with reference to the detailed description and appended figures in which:

Fig. 1 depicts a functional block diagram of an embodiment of a processor chip within which embodiments of the present invention may find application;

Fig. 2 depicts a functional block diagram of a data busing scheme for use in a processor, which has a microcontroller and a digital signal processing engine, within which embodiments of the present invention may find application;

Fig. 3 depicts a functional block diagram of a processor logic configuration for multiplying two maximally negative 16-bit fractional numbers according to embodiments of the present invention; and

Fig. 4 depicts a method of multiplying two maximally negative 16-bit fractional numbers according to embodiments of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

According to embodiments of the present invention, methods and processors for multiplying two maximally negative fractional numbers to produce a 32-bit result are provided. This type of multiplication may be executed using a 16-bit DSP multiplier and produce a 32-bit result. The identification of a multiplication operation employing two maximally negative 16-bit fractional numbers enables manipulation of processing to correct a maximally negative result and produce a maximally positive result. Negate logic with a control block examines results produced by the 16-bit DSP multiplier and determines whether there are a combination of bits signifying the multiplication of two maximally negative 16-bit fractional numbers. The determination of the required bit combination initiates negate processing for correcting the results. This type of multiplication operation utilizes of a 16-bit DSP multiplier to produces accurate 32-bit results when the multiplication of two maximally negative fractional number occurs as well as reduces the overall cost of the processor.

In order to describe embodiments of multiplying two maximally negative fractional numbers, an overview of pertinent processor elements is first presented with reference to Figs. 1 and 2. The multiplication of two maximally negative fractional numbers is then described more particularly with reference to Figs. 3-4.

Overview of Processor Elements

Fig. 1 depicts a functional block diagram of an embodiment of a processor chip within which the present invention may find application. Referring to Fig. 1, a processor 100 is coupled to external devices/systems 140. The processor 100 may be any type of

processor including, for example, a digital signal processor (DSP), a microprocessor, a microcontroller or combinations thereof. The external devices 140 may be any type of systems or devices including input/output devices such as keyboards, displays, speakers, microphones, memory, or other systems which may or may not include processors. Moreover, the processor 100 and the external devices 140 may together comprise a stand alone system.

The processor 100 includes a program memory 105, an instruction fetch/decode unit 110, instruction execution units 115, data memory and registers 120, peripherals 125, data I/O 130, and a program counter and loop control unit 135. The bus 150, which may include one or more common buses, communicates data between the units as shown.

The program memory 105 stores software embodied in program instructions for execution by the processor 100. The program memory 105 may comprise any type of nonvolatile memory such as a read only memory (ROM), a programmable read only memory (PROM), an electrically programmable or an electrically programmable and erasable read only memory (EPROM or EEPROM) or flash memory. In addition, the program memory 105 may be supplemented with external nonvolatile memory 145 as shown to increase the complexity of software available to the processor 100. Alternatively, the program memory may be volatile memory which receives program instructions from, for example, an external non-volatile memory 145. When the program memory 105 is nonvolatile memory, the program memory may be programmed at the time of manufacturing the processor 100 or prior to or during implementation of the processor 100 within a system. In the latter scenario, the processor 100 may be programmed through a process called in-line serial programming.

The instruction fetch/decode unit 110 is coupled to the program memory 105, the instruction execution units 115 and the data memory 120. Coupled to the program memory 105 and the bus 150 is the program counter and loop control unit 135. The instruction fetch/decode unit 110 fetches the instructions from the program memory 105 specified by the address value contained in the program counter 135. The instruction fetch/decode unit 110 then decodes the fetched instructions and sends the decoded instructions to the appropriate execution unit 115. The instruction fetch/decode unit 110 may also send operand information including addresses of data to the data memory 120 and to functional elements that access the registers.

The program counter and loop control unit 135 includes a program counter register (not shown) which stores an address of the next instruction to be fetched. During normal instruction processing, the program counter register may be incremented to cause sequential instructions to be fetched. Alternatively, the program counter value may be altered by loading a new value into it via the bus 150. The new value may be derived based on decoding and executing a flow control instruction such as, for example, a branch instruction. In addition, the loop control portion of the program counter and loop control unit 135 may be used to provide repeat instruction processing and repeat loop control as further described below.

The instruction execution units 115 receive the decoded mathematical instructions from the instruction fetch/decode unit 110 and thereafter execute the decoded mathematical instructions. As part of this process, the execution units may retrieve a set of source operands via the bus 150 from data memory and registers 120. During instruction processing, such as mathematical operation instructions, the set of source operands may be

fetched from data memory and registers 120 as specified in the mathematical operation instructions. The set of set of source operands may be transferred from the data memory and registers 120 to registers prior to delivery to the execution units for processing. Alternatively, the set of source operands may be delivered directly from the data memory and registers 120 to the execution units for processing. Execution units may also produce outputs to registers and/or the data memory 120. The execution units may include an arithmetic logic unit (ALU) such as those typically found in a microcontroller. The execution units may also include a digital signal processing engine including a multiply and accumulate unit (MAC), a floating point processor, an integer processor or any other convenient execution unit. A preferred embodiment of the execution units and their interaction with the bus 150, which may include one or more buses, is presented in more detail below with reference to Fig. 2.

The data memory and registers 120 are volatile memory and are used to store data used and generated by the execution units. The data memory 120 and program memory 105 are preferably separate memories for storing data and program instructions respectively. This format is a known generally as a Harvard architecture. It is noted, however, that according to the present invention, the architecture may be a Von-Neuman architecture or a modified Harvard architecture which permits the use of some program space for data space. A dotted line is shown, for example, connecting the program memory 105 to the bus 150. This path may include logic for aligning data reads from program space such as, for example, during table reads from program space to data memory 120.

Referring again to Fig. 1, a plurality of peripherals 125 on the processor may be coupled to the bus 125. The peripherals may include, for example, analog to digital converters, timers, bus interfaces and protocols such as, for example, the controller area network (CAN) protocol or the Universal Serial Bus (USB) protocol and other peripherals. The peripherals exchange data over the bus 150 with the other units.

The data I/O unit 130 may include transceivers and other logic for interfacing with the external devices/systems 140. The data I/O unit 130 may further include functionality to permit in circuit serial programming of the Program memory through the data I/O unit 130.

Fig. 2 depicts a functional block diagram of a data busing scheme for use in a processor 100, such as that shown in Fig. 1, which has an integrated microcontroller arithmetic logic unit (ALU) 270 and a digital signal processing (DSP) engine 230. This configuration may be used to integrate DSP functionality to an existing microcontroller core. Referring to Fig. 2, the data memory 120 of Fig. 1 is implemented as two separate memories: an X-memory 210 and a Y-memory 220, each being respectively addressable by an X-address generator 250 and a Y-address generator 260. The X-address generator may also permit addressing the Y-memory space thus making the data space appear like a single contiguous memory space when addressed from the X address generator. The bus 150 may be implemented as two buses, one for each of the X and Y memory, to permit simultaneous fetching of data from the X and Y memories.

The W registers 240 are general purpose address and/or data registers. The DSP engine 230 is coupled to both the X and Y memory buses and to the W registers 240. The DSP engine 230 may simultaneously fetch data from each the X and Y memory, execute

instructions which operate on the simultaneously fetched data and write the result to an accumulator (not shown) and write a prior result to X or Y memory or to the W registers 240 within a single processor cycle.

In one embodiment, the ALU 270 may be coupled only to the X memory bus and may only fetch data from the X bus. However, the X and Y memories 210 and 220 may be addressed as a single memory space by the X address generator in order to make the data memory segregation transparent to the ALU 270. The memory locations within the X and Y memories may be addressed by values stored in the W registers 240.

Any processor clocking scheme may be implemented for fetching and executing instructions. A specific example follows, however, to illustrate an embodiment of the present invention. Each instruction cycle is comprised of four Q clock cycles Q1 – Q4. The four phase Q cycles provide timing signals to coordinate the decode, read, process data and write data portions of each instruction cycle.

According to one embodiment of the processor 100, the processor 100 concurrently performs two operations – it fetches the next instruction and executes the present instruction. Accordingly, the two processes occur simultaneously. The following sequence of events may comprise, for example, the fetch instruction cycle:

Q1: Fetch Instruction
Q2: Fetch Instruction
Q3: Fetch Instruction
Q4: Latch Instruction into prefetch register, Increment PC

The following sequence of events may comprise, for example, the execute instruction cycle for a single operand instruction:

Q1: latch instruction into IR, decode and determine addresses of operand data

Q2: fetch operand
Q3: execute function specified by instruction and calculate destination address for data
Q4: write result to destination

The following sequence of events may comprise, for example, the execute instruction cycle for a dual operand instruction using a data pre-fetch mechanism. These instructions pre-fetch the dual operands simultaneously from the X and Y data memories and store them into registers specified in the instruction. They simultaneously allow instruction execution on the operands fetched during the previous cycle.

Q1: latch instruction into IR, decode and determine addresses of operand data
Q2: pre-fetch operands into specified registers, execute operation in instruction
Q3: execute operation in instruction, calculate destination address for data
Q4: complete execution, write result to destination

Maximally Negative Fractional Number Multiplication

Fig. 3 depicts a functional block diagram of a processor for multiplying two maximally negative fractional numbers to produce a 32-bit maximally positive fractional result according to an embodiment of the present invention. Referring to Fig. 3, the processor includes data memory 305 for storing data, such as two maximally negative fractional operands, used and generated by the processor. The processor also includes registers 305 for containing data that the processor generates and employs during processing mathematical operation instructions. The registers 305 may include a set of 16-bit W registers defined for mathematical operation instructions. The set of W registers may contain data including an operand and an effective address of an operand in data memory 300. The effective address of an operand in data memory 305 can be specified as an

address or an address plus an offset. The processor also includes DSP unit 310, negate logic 325, fractional alignment logic 330, sign extension logic 335 and accumulator 340.

The DSP unit 310 can include registers 315 that receive operands from the registers 305 and the data memory 300. The DSP unit 310 includes DSP logic 320, which can receive inputs from the registers 315 and produces a 32-bit maximally negative result output to the fractional alignment logic 330 and the two most significant bits of the maximally negative result output to the negate control logic with control block 325. The DSP logic 320 includes a 16-bit multiplier that executes multiplicative and logic operations on operands fetched from the registers 305 and the data memory 300.

DSP logic 320 is activated upon the execution of mathematical operation instructions. In this regard, when a mathematical operation instruction is executed control signals cause the DSP unit to fetch operands from the registers 305 and the data memory 300. The control signals also cause the DSP logic 320 to operate on the fetched operands to produce result outputs in accordance with the instruction. The result outputs depend upon the instruction executed and the source operands. The result outputs may correspond to a maximally negative result output. The production of a maximally negative result output is based on the multiplication of two maximally negative numbers, such as –1 * -1. After generating the result outputs, the DSP unit 310 writes the result outputs into the correct registers 305, data memory 300, and accumulator 340.

The fractional alignment logic 330 can receive result outputs produced by DSP unit 310 and produces modified result outputs. The result outputs may be 32-bits in length. The 32-bit representation of result outputs are shifted in the direction of the most

significant bit by one bit to discard the most significant bit, while a zero is shifted/inserted in as the least significant bit to produce the modified result outputs.

Negate logic 325 can receive modified result outputs and the two most significant bits of bits representing result outputs as well as produce a maximally positive result output. Negate logic 325 includes a control block which can detect whether result outputs correspond to maximally negative results to generate a control signal to modify a negate control signal. The modification of the negate control signal enables negate logic 325 to correct maximally negative results to produce maximally positive results. The detection of a maximally negative result indicates that the operands operated on during the multiplication operation that produces the maximally negative result are two maximally negative fractional numbers, such as –1 * -1.

Control block examines the two most significant bits of the bits representing the result output and determines whether the two most significant bits in the set of bits representing the result output have a particular bit combination. The two most significant bits in the set of bits examined are the thirtieth and thirty-first bits for the set of bits representing the result output. Upon determining that the thirtieth and thirty-first bits are one and zero respectively, control block generates a control signal to modify a negate control signal generated by negate logic. The modified negate signals causes negate logic to perform a two's compliment operation on the result output correcting the maximally negative result output to a maximally positive result output. The error introduce by correction of the maximally negative result output is nominal, and more, specifically one least significant bit of the result output.

Sign extension logic 335 receives result outputs, including maximally positive result outputs, and produces sign extended result outputs. Result outputs are extended from a length of 32-bits to a length of 40-bits. Accumulators 340 accumulate result outputs that have been sign extended by sign extension logic 335. The Accumulators may be two 40-bit accumulators.

Fig. 4 depicts a method of multiplying two maximally negative fractional numbers, such as -1 * -1, to produce a 32-bit maximally positive fractional result and is best understood when viewed in combination with Fig. 3. Referring to Fig. 4, in step 400, a DSP multiplier 320 of a DSP unit 310 for a processor fetches operands from source locations. In the embodiment of Fig. 4, one operand is fetched from data memory 300, while the other operand is fetched from a register 305. Each of the operands may be a maximally negative fractional number. In step 410, the DSP multiplier 320 of the processor may execute a multiplication operation using the operands to produce a result output from the DSP unit 310. The result output may correspond to a maximally negative result. Maximally negative results are produce when two maximally negative numbers, such as -1, are multiplied.

In step 420, fractional alignment logic 330 shifts result output produced by DSP unit 310 in the direction of the most significant bit in a set of bit representing the result output and inserts a zero at the least significant bit in the set of bits representing the result output. In step 430, negate logic receives a modified result output produced by fractional alignment logic 330 and the two most significant bits of a set of bits representing the result output produced by DSP unit 310. Negate control logic 325 determines whether the result output corresponds to a maximally negative result. If the result output corresponds to a

maximally negative result the method proceeds to step 440, otherwise the method proceeds to step 450. In step 440, the result output is corrected from a maximally negative result to a maximally positive result. In the Fig. 4 embodiment, in step 450, the result output is sign extended by sign extension logic 335. One having skill in the art would recognize that the result output may be sign extended at any point after step 420. In step 460, an accumulator accumulates the sign extended result output.

While specific embodiments of the present invention have been illustrated and described, it will be understood by those having ordinary skill in the art that changes may be made to those embodiments without departing from the spirit and scope of the invention.